浙江大学

Fundamentals of Data Structures

Project2

Project Report

2020~2021 秋冬学期 2020 年 11 月 30 日

Dijkstra Sequence

Categories

Project2

Chapter 1: Introduction (6 pts.)

1.1 Background and significance of topic selection

Dijkstra's algorithm is one of the very famous greedy algorithms. It is used for solving the single source shortest path problem which gives the shortest paths from one particular source vertex to all the other vertices of the given graph. It was conceived by computer scientist Edsger W. Dijkstra in 1956 and published three years later.

In this algorithm, a set contains vertices included in shortest path tree is maintained. During each step, we find one vertex which is not yet included and has a minimum distance from the source, and collect it into the set. Hence step by step an ordered sequence of vertices, let's call it **Dijkstra sequence**, is generated by Dijkstra's algorithm.

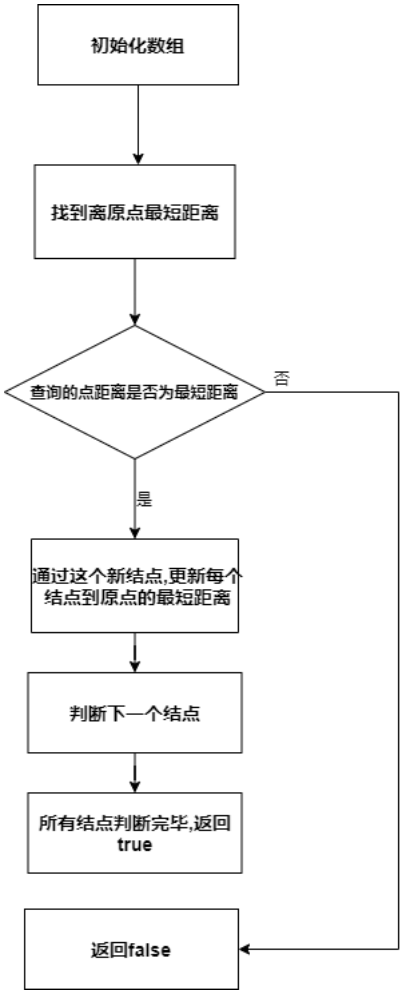
On the other hand, for a given graph, there could be more than one Dijkstra sequence.

1.2 Our goals

From the input information, we need to judge whether or not each of the queries is a Dijkstra sequence or not.
Input :
The first line contains 2 positive integers Nv (≤10^3) and Ne (≤10^5), which are the total numbers of vertices and edges, respectively.
Then Ne lines follow, each describes an edge by giving the indices of the vertices at the two ends, followed by a positive integer weight (≤100) of the edge.
Finally input the number of queries, K (1≤K≤100),
Then K lines follow, each contains a permutation(排列) of the Nv vertices. The first vertex is the root in Dijkstra sequence.
Output Specification:
For each of the K sequences, print in a line Yes if it is a Dijkstra sequence, or No if not.

Chapter 2: Algorithm Specification (12 pts.)

2.1 Overall architecture design



2.2 Algorithm design

We use a “for” loop to traverse all vertices,
For each given query vertex:
first off, we need find the min distance in distance array, if the min distance isn’t identical the given vertex’s distance, we will know this given sequence isn’t a Dijkstra sequence. Otherwise, we write this vertex into log.
Then, we update all vertex’s distance from the root in distance array.
Finally, we judge next given vertex.

2.3Main data structures

```
int Nv, G[1010][1010] = {0}, distance[1010], query[1010];
```

distance array saves the distance from the root to the vertex;
G is the Graph which saved the weighted edges.
Query array saves the given queries.
int log[1010] = {0};// first off, all points have not been traversed.

Chapter 3: Testing Results (20 pts.)

test case	Actual behavior of my program
5 7 1 2 2 1 5 1 2 3 1 2 4 1 2 5 2 3 5 1 3 4 1 1 5 1 3 4 2 Normal test1	**YES**
5 7 1 2 2 1 5 1 2 3 1 2 4 1 2 5 2 3 5 1 3 4 1 1	**YES**

5 3 1 2 4 Normal test2	
5 7 1 2 2 1 5 1 2 3 1 2 4 1 2 5 2 3 5 1 3 4 1 1 2 3 4 5 1 Normal test3	**YES**
5 7 1 2 2 1 5 1 2 3 1 2 4 1 2 5 2 3 5 1 3 4 1 1 1 5 3 2 4 My designed test4	**YES**
5 7 1 2 2 1 5 1 2 3 1 2 4 1 2 5 2	**NO**

3 5 1 3 4 1 1 3 2 1 5 4 ， root 3 to third vertex 1 , distance d[1] is 3（3->2->1），but the min is 1（3->4）.	
5 7 1 2 2 1 5 1 2 3 1 2 4 1 2 5 2 3 5 1 3 4 1 1 0 0 0 0 0 Test empty	**NO**
2 1 1 2 1 1 1 2 the cases with the smallest sizes	**Yes**

Chapter 4: Analysis and Comments (10 pts.)

time complexities of the algorithms :O(V^2)

because

int Dijkstra(int root);// Two loops nested, each repeated V times

space complexities of the algorithms

The all space complexities is Θ(n^2)

Because G[1001][1001] save the graph

2. If implemented with an adjacency list, in the worst case, the time complexity of O(n*m) seems to be larger than n², but the space complexity will change from n ^2 to m, which is much less than now space complexity.

3. After heap optimization, Dijkstra's time complexity can reach nlogn. If the graph is particularly dense, that is, m is particularly large (for example, the complete graph is n^2), then nlogn is smaller than m.

How to optimize heap optimization? Observe the above code, each time a loop is nested to find the minimum dis value. Here, we can use a priority queue. Whenever a new point is searched, it will be thrown into the priority queue, so that the top of the queue is absolutely optimal every time. It can be a further possible improvements.

In the process, I meet lots of bugs. I use `printf and debug->step in, step over` to debug. It really exam my attention and patience. Actually many problem occurs in whether or not if include a Statement block.

We could typedef enum{false,true} bool; define MAX

Appendix: Source Code

```
#include <stdio.h>
#include <stdlib.h>
```

```
#include <math.h>
#define MAX(a,b) (a>b)?a:b
//Dijkstra 每次找目前距原点最近的点，若有几个距离相同的会取其中一个，但其实先检查另外几个点也可
// distance save the distance from the node to root;
const int INF = 0x3fffffff;// min 记录最小值，一开始设置为最大.
int Nv, G[1010][1010] = {0}, distance[1010], query[1010];
int Dijkstra(int root){
    int i,min = INF, j,tmp;
    for(i = 0;i< 1010;i++){
    distance[i] = INF; //一开始到所有点距离置为最大.
  }
  int log[1010] = {0};// 所有的点都没有遍历过.
  distance[root] = 0; // 原点到原点的距离是 0
  //printf("init finished \n");
  for(i = 0; i < Nv; i++){   //如果查询的每个点都没问题那就返回 1
      min = INF;//一开始最短距离置为最大.
  // printf("begin find min \n");
      for(j = 1;j <= Nv;j++){
          if(log[j] == 0 && distance[j] < min){   //如果没遍历过，而且到原点的距离更短
```

```
                min = distance[j];          //那就更新 最短距离
        //      printf("%d\n",min);
            }
        }
  //      printf("search min finished\n");//如果查询的节点query[i]到原点的距离是 min
  //      printf("min = %d j = %d Nv = %d query[0] = %d\n",min,j,Nv,query[i]);//query[i] ,Nv 输出没有问题.
        // Nv  =5 is correct, query[0] = 5  is correct ,
        if(distance [query[i]]  == min ){  // 如果查询的点,最短距离是我们找到的最短距离.
            tmp = query[i] ; //记录下查询的点
    //      printf("%d\n",tmp);
        }
        else{
            return 0; //此 query 非所要求的 Dijkstra sequence，返回 false
        }
        //这个点已经查过了. 可以用 0 和 1 来标记, 1 表示记过了 static vis[1001]
        if(!log[tmp]){
            log[tmp] = 1 ;// 遍历过了,记录为 1
```

```
        }
    //  printf("begin update distance\n");
        // 没有遍历过这个点&& 有这个边 &&如果他到原点距离> 他到查询节点+查询节点到原点距离
        //更新每一个结点到原点的距离
        for(j = 1 ; j < Nv+1;j++){
            if( log[j] == 0    &&   G[tmp][j] != 0 && distance[j] > distance[tmp] + G[tmp][j]      ){
                distance[j]    =    distance[tmp]    +    G[tmp][j];
    //update distance from j to root;
        //      printf("%d\n",j);
            }
        }
    }
    return 1 ;//如果查询的每个点都没问题那就返回 1
  }
  int main(){
       int   Ne, v,u, distance, K,i,j,judge;
      scanf("%d%d", &Nv, &Ne);//输入顶点数和边数
  //      Nv = 5, Ne = 7;//初始化一些值方便调试
  //G[1][2] = G[2][1] = 2;
  //G[1][5] = G[5][1] = 1;
```

```
    //G[2][3] = G[3][2] = 1;
    //G[2][4] = G[4][2] = 1;
    //G[2][5] = G[5][2] = 2;
    //G[3][5] = G[5][3] = 1;
    //G[3][4] = G[4][3] = 1;
        for (i = 0; i < Ne; i++){   //输入每条边的大小
            scanf("%d%d%d", &u, &v, &distance);//输入距离
            G[u][v] = G[v][u] = distance;//  图是没有[0][0]的
        }
        scanf("%d",&K);//输入查询语句个数
     for (i = 0; i < K; i++){ //查询 K 次
            for (j = 0; j < Nv; j++) { //每条语句有 Nv 个,也就是顶点个数
            scanf("%d", &query[j]);//输入查询的每个点
        }
        judge = Dijkstra(query[0]); //  传入原点
            printf("%s\n", judge ? "Yes" : "No");//judge 为 1 就输出 yes ,为 0 输出 no
        }
        return 0;
    }
```

Declaration

I hereby declare that all the work done in this project titled " Dijkstra Sequence" is of my independent effort.